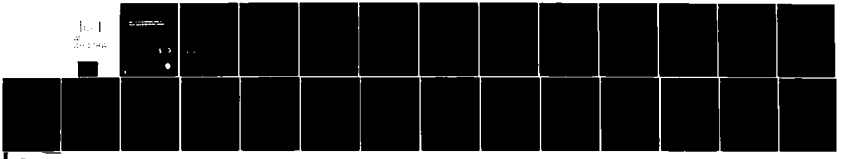
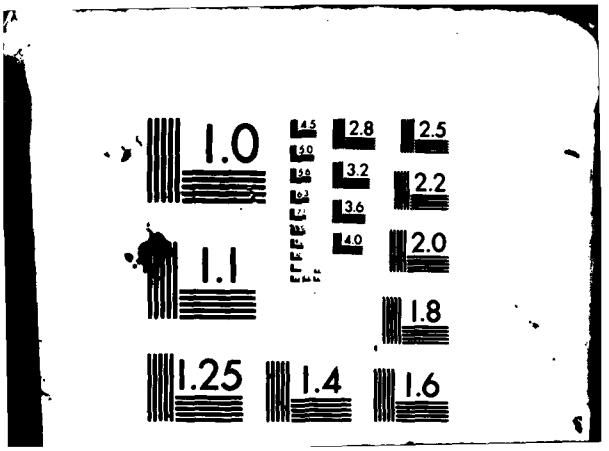
AD-A114 788 SYSTEMS RESEARCH LABS INC DAYTON OH F/G 9/2
MANX1 A SYSTEM FOR COMPUTERIZED CONTROL OF AND DATA ACQUISITION--ETC(U)
MAR 82 D W BLICK, J T YATES, T G WHEELER F33615-80-C-0603
UNCLASSIFIED SAM-TR-82-6 NL



END
DATE FILMED
6 82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
10.0

AD A114788

Report SAM-TR-82-8

[redacted]: A SYSTEM FOR [redacted] CONTROL OF [redacted] EXPERIMENTS

[redacted]
J. Terry Tritt, Ph.D.
Thomas G. Wheeler, Ph.D.
[redacted]
[redacted]
Dayton, Ohio [redacted]

March 1982

[redacted]

[redacted]

[redacted]

NOTICES

This final report was submitted by Systems Research Laboratories, Inc., 2800 Indian Ripple Road, Dayton, Ohio 45440, under Contract F33615-[illegible], Job Order 7757-05-[illegible], with the USAF School of Aerospace Medicine, Aerospace Medical Division, AFSC, Brooks Air Force Base, Texas. Captain T. E. Dayton (SAM/RZW) was the Laboratory Project Scientist-in-Charge.

When U.S. Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The animals involved in this study were procured, [illegible] and used in accordance with the Animal Welfare Act of 1970 and the "Guide for Care and Use of Laboratory Animals" prepared by the Institute of Laboratory Animal Resources - National Research Council.

This report has been reviewed by the Office of Public Affairs [illegible] and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Thomas E. Dayton

THOMAS E. DAYTON, Captain, USAF
Project Scientist

Donald R. Thomas

DONALD R. THOMAS, Ph.D.
Supervisor

R L Dehart

ROY L. DEHART
Colonel, USAF, MC
Commander

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: SAM-TR-82-6	2. GOVT ACCESSION NO.: AD-A114788
	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE *(and Subtitle)*: MANX: A SYSTEM FOR COMPUTERIZED CONTROL OF AND DATA ACQUISITION FROM BEHAVIORAL EXPERIMENTS	5. TYPE OF REPORT & PERIOD COVERED: Final Report August 1979 - June 1980
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Dennis W. Blick, Ph.D. J. Terry Yates, Ph.D. Thomas G. Wheeler, Ph.D.	8. CONTRACT OR GRANT NUMBER(s): F33615-80-C-0603
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Systems Research Laboratories, Inc. 2800 Indian Ripple Road Dayton, Ohio 45440	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62202F 7757-05-43
11. CONTROLLING OFFICE NAME AND ADDRESS: USAF School of Aerospace Medicine (RZW) Aerospace Medical Division (AFSC) Brooks Air Force Base, Texas 78235	12. REPORT DATE: March 1982
	13. NUMBER OF PAGES: 23
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)*: Approved for public release; distribution unlimited.	
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*	
18. SUPPLEMENTARY NOTES	
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*: Behavior Instrumentation Computer control Minicomputers	
20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*: This report is a general description of MANX, an integrated hardware/software system for the precise control and measurement of animal behavior. MANX and its State Programming Language facilitate optimal utilization of the NOVA laboratory computer for behavioral experiments.	

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

MANX: A SYSTEM FOR COMPUTERIZED CONTROL OF AND DATA ACQUISITION FROM BEHAVIORAL EXPERIMENTS

This report will describe the functional properties of MANX (MANX Systems, Inc.), an integrated software and hardware system that provides a means for using a NOVA computer (Data General Corporation) to precisely control and acquire data from a variety of behavioral experiments.

Because of its unique applicability to and advantages for the types of research being conducted there, MANX was implemented at the USAF School of Aerospace Medicine (USAFSAM) by Systems Research Laboratories. MANX enables the available laboratory computer to simultaneously control and collect data from as many as 30 distinct and independent experiments. This provides the capability for several investigators to conduct multiple behavioral research projects within the confines of a single laboratory. The simple yet powerful MANX programming language allows rapid generation and modification of programs to control a variety of experiments. Flexible input/output (I/O) arrangements facilitate interfacing the computer to diverse experimental apparatus.

BACKGROUND

The MANX software system is basically an adaptation of SKED (State Systems, Inc.) (12). SKED was developed for use with the Digital Equipment Corporation's PDP-8 computers. MANX, which was modeled after SKED, was developed to provide the advantages of SKED to users of NOVA computers. Since MANX and SKED are very similar functionally, much of the following description applies equally to both software systems.

The development of specialized software packages for laboratory use in the behavioral sciences was a natural consequence of two historical trends. First, as empirical knowledge and theory in the area grew, experimental questions grew more precise and detailed. This required more and more precise instrumentation and data acquisition. Automation of experiments has several obvious advantages. A mechanical experimenter not only removes much of the drudgery from collecting large numbers of observations, it is also more accurate and reliable, it presents each subject with the same experimental situation, it doesn't display biases associated with fondness for particular experimental hypotheses, and it will work 24 hours a day if the experiment requires it.

In the late 1930s, Skinner (11) introduced new instrumental technologies and a new empirical approach into the behavioral laboratory. This resulted in a new school of psychological thought, known as the Experimental Analysis of Behavior, which concentrated on describing the behavioral effects of a variety of stimulus-response contingencies. In 1957, Ferster and Skinner (1) published a monumental handbook, defining a large variety of schedules of reinforcement and describing their effects on operant behavior. By this time, many behavioral experiments entailed the use of large racks of electromechanical devices (relays, timers, counters, cumulative recorders, etc.) for stimulus control and response recording purposes.

During the late 1950s and early 1960s, as transistor-based electronic technology developed, many of the older electromechanical devices were gradually replaced by electronic devices packaged as digital logic modules. Although the new electronic devices were smaller, faster, and more reliable than the electromechanical devices they replaced, they did not modify the basic nature of experimental control. Each new experiment or modification of experimental design required that the apparatus be "programmed" by changing the "hard-wired" relationships among the many components. The design, construction, testing, and modifications of these hard-wired programs was a tedious process, at best.

As small digital computers became available for laboratory use, experimental psychologists were quick to recognize the advantages of this new technology for experimental control and analysis (9,15,16). The speed, flexibility, and computational power of the computer greatly increased the variety of experiments that could be conducted in each laboratory and made possible new classes of experiments that were impossible or unfeasible with the older technologies. The major advantage, however, was that computers allowed modification or generation of experiments by purely symbolic (software) manipulations, without requiring extensive rewiring of the apparatus. In other words, once the computer was interfaced to one or several experimental stations in such a way that it could sense responses and operate stimuli, any number of different experiments could be performed on the station(s) simply by running different programs in the computer.

Initially the computer programs generated to control behavioral experiments tended to be in the machine language of the particular computer in use in each laboratory. This made the programming of changes in experiments almost as tedious as by the old method (rewiring). As laboratory computers grew larger (in terms of memory capacity) and more powerful, programs in higher level languages (e.g., BASIC, FORTRAN, APL) were developed. However, these languages are specialized for computational procedures rather than the on-line control of processes in real-time. Also, these languages typically use computer memory inefficiently. Consequently, these languages were cumbersome to use for behavioral programming, which involves control sequences that vary depending on the behavior of the experimental subject. There was thus a clear need for higher level programming language systems specialized to handle the problems of behavior analysis and control.

A number of systems were developed to meet this need (17). The most widely known of the software packages specialized for behavioral control are SCAT (State Change Algorithm Translator, developed and distributed by Grason-Stradler, Inc. (10)); INTER-ACT (an Interactive Automated Contingency Translator, developed and distributed by BRS-LVE--now BRS-Foringer, Inc. (8)); and SKED (developed in an academic setting and distributed by a nonprofit corporation, State Systems, Inc.). Because of its flexibility, efficient hardware utilization, and relatively low-cost implementation, SKED is currently the most widely used of these systems.

SKED was originally developed in 1966. It has been expanded and refined to take advantage of mass storage devices and more sophisticated operating systems. In current implementations (PDP-8s with 32K words of memory and disk storage devices), it can simultaneously control 12 independent behavioral stations in a time-sharing mode. This allows program development or data processing to occur while the behavior control system is functioning. SKED has

not yet been adapted to operate in the environment of more modern Digital Equipment Corp. machines; e.g., the PDP-11.

The MANX system was developed to meet the behavioral programming needs of users of NOVA computers. It was modeled after SKED but has minor functional differences. For example, MANX can control up to 30 independent stations, as compared to SKED's 12. On the other hand, MANX requires 64K words of memory in order to operate in a time-sharing mode. Since the NOVA 800 computer in our laboratory has only 32K words of memory, the machine is fully dedicated to the behavioral control task whenever any behavioral station is in operation. Off-line data analysis and program development must be accomplished during periods when no behavioral stations are in use.

MANX SYSTEM DESCRIPTION

Two brief published descriptions of the MANX system (originally called NOVA SKED) are available (5,6). Complete documentation is available in the form of system manuals provided by the vendor (2-4). The purpose of this paper is to provide a condensed functional description of the system as implemented at USAFSAM, emphasizing the features that have proven particularly useful for studying hazardous-environment effects on animal performance models.

The MANX system consists of two major components: software systems and hardware components. The hardware components include two digital I/O cards in the computer and two interface panels in the laboratory. The I/O cards and interface panels allow the computer to sense the subjects' responses and to operate apparatus that present stimuli to the subjects. Sixty-four input (response) lines and 128 output (stimulus) lines are available. The specific connections of these hardware components for individual experiments will be documented in detail in separate reports for each major experiment. This report will concentrate on the MANX software system, describing only in general terms the hardware environment in which it operates at USAFSAM.

Figure 1 is a simplified block diagram of the systems involved in experimental control, data acquisition, and data analysis, when MANX is used on the NOVA 800 computer. The dashed vertical line in Figure 1 separates two distinct but interacting functional systems. The A side of the figure depicts the components involved in on-line control of experimental operations, including data acquisition. In the current system configuration, operations on the A side preclude operations on the B side. This means that program generation and modification, as well as data analysis, must be done off-line, when the MANX runtime system (RTS) is deactivated.

All computer operations are under the executive control of RDOS, Data General Corporation's real-time disk operating system. This operating system is fully documented in Data General manuals. RDOS provides for operation of the system's real-time clock, interpretation of commands from the system console, analog-to-digital and digital-to-analog conversion, input and output operations on the digital I/O, and manipulation of data and text files on the system's mass storage devices (disks and magnetic tape).

Included with RDOS is a text editor routine which is used to enter and modify source programs for experimental control and data analysis. As indicated

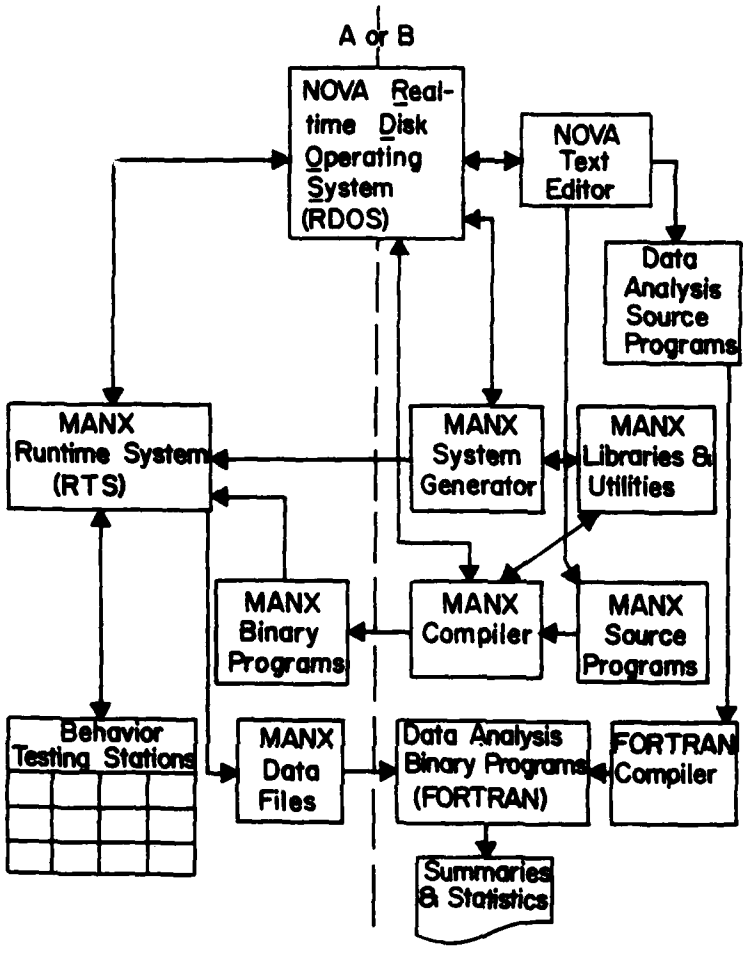


Figure 1. MANX Software System Block Diagram

A. MANX RTS operates under RDOS. RTS is produced by the MANX system generation (B). When RTS is in operation, MANX binary programs are loaded for the control of behavioral testing stations. RTS accumulates and stores data from the stations.

B. MANX users generate source programs, using the NOVA editor. MANX source programs are translated into machine language by the MANX compiler. Data analysis source programs are translated to machine language by the FORTRAN compiler. Data analysis programs operate on MANX data files to sort, collate, and summarize the outcomes of experiments.

in Figure 1(B), source programs are of two distinct types. MANX source programs are written in State Logic Notation. Data analysis programs are written in FORTRAN IV, a widely used higher level language provided and documented by Data General for use on this computer.

Four major components of the MANX software system are indicated in Figure 1(B): the system generator, library and utility package, compiler, and source programs.

The MANX system generator is a program to generate a MANX RTS tailored to the facilities and needs of the individual laboratory. The program elicits keyboard inputs from the user to determine number of behavioral stations (1-30), characteristics of the computer system (hardware or software multiply/divide functions, types of I/O devices), and which MANX system and user functions (described later) are to be included in the RTS. This allows the user to tailor a runtime system that uses the smallest amount of computer memory possible, consistent with experimental requirements. For a given laboratory, system generation would typically be performed only once unless experimental requirements or hardware configuration changed substantially. Minor changes, such as modification of station input/output configurations, can be made easily without going through the system generation process.

At system generation time, the MANX system generator searches through the contents of the MANX library and utility package so as to provide the runtime system with the capabilities required by the user. This library includes all FORTRAN and assembler programs and subroutines required for experimental control and data acquisition, as well as a library of test routines and data analysis programs. It also provides users with easy-to-use programs for mass-storage backup (on magnetic tape) of experimental data. Routines for decoding, sorting, and statistically reducing and plotting experimental data are also provided. User-generated data analysis routines, written in FORTRAN, can readily use MANX library subroutines to sort, decode, and manipulate MANX data files.

The MANX compiler (MXCOM) is a large FORTRAN program that accepts MANX source programs as inputs and produces MANX system binary programs which can be called and run by the MANX RTS (Fig. 1(A)). MANX source programs are written in State Logic Notation, a simple yet powerful language that is easily learned and used by behavioral experimenters. (This language will be described in detail later.) The MANX binary programs produced by MXCOM can be loaded and run by MANX RTS in any behavioral station under MANX control.

MXCOM is a two-pass compiler, with an optional third pass to produce listing files. On the first pass, MXCOM scans the source program for syntax errors. If and when such errors are found, detailed error messages presented on the console allow the user to identify and correct the errors quickly. If syntax errors are found, MXCOM terminates after the first pass. In the second pass, MXCOM produces the binary programs that will actually control experiments when they are loaded by the RTS. The listing files, which can be produced by an optional third pass, are generally useful only to systems analysts.

The MANX RTS (Fig. 1(A)) is a multitasking assembly language program capable of running in the foreground or background under RDOS. The RTS can control as many as 30 experimental stations operating simultaneously. Each station

operates independently of all others. With MANX binary programs stored on the disk, any station can be loaded, started, or stopped while other stations are in operation, using the same or different MANX binary programs. A disk data file for each station is automatically opened when it is loaded, and closed when it is stopped. Relative input and output bit assignments in the MANX binary programs are resolved by the RTS to absolute inputs and outputs for each station by reference to a look-up table in a disk file (MANX.IO) which the user prepares in advance. Alternative I/O assignments can be generated, and the RTS will refer to any I/O look-up table the user specifies when starting the RTS. This provides great flexibility in connecting the interface to experimental apparatus, as changes in these connections require changes only in the I/O look-up table, not in the programs.

The RTS responds to a variety of keyboard commands. These allow, for example, examining experimental data (values of counters and program variables) as it accumulates; modifying I/O assignments or program variables "on-the-fly"; determining which stations are active and what program is running in each active station; effecting start, stop, pause, and resume for any station; artificially turning any input or output line on or off by keyboard command; and checking on memory and/or disk space availability.

The RTS is a real-time operating system that derives its timing functions from the RDOS system clock. The RTS can accommodate clock rates of 1, 10, 100, or 1000 Hz; system overhead, however, tends to become excessive at the highest clock rate. The current system uses a clock rate of 100 Hz, which provides temporal resolution of 10 msec for response detection or stimulus presentation.

The RTS stores data from each station in one of two 256-word buffers in memory. When one of these buffers is filled, it is automatically written to the disk data file that was opened by the RTS when the station was loaded. Data accumulates in the second buffer while the first is being transferred to disk. The active data buffer is written to the disk data file when the station stops normally at the end of the experimental run.

For each experimental run, the disk data file contains header information that includes date, experiment number, subject number, group number, program number, and number of counters. Experiment, subject, and group numbers are supplied by the operator when loading; the other information is included automatically. At the time of system generation, the user has the option of including treatment, dose, and operator identification in the data-set header.

MANX data can be gathered and stored in two basic forms: the relay-rack form (counters) and/or as coded interevent times (IET). The relay-rack form is a vestige of the time when most experimental analyses of behavior were performed using racks of electromechanical equipment (relay racks) to control the experiment and record the outcomes. A bank of counters was usually somewhere in the relay rack. Each counter accumulated the total number of occurrences of some class of events of interest. At the end of each experiment (or sometimes at intervals during the course of an experiment), the values of the counters were recorded and the counters were reset to zero. MANX provides a similar facility. As many as 4095 distinct classes can be counted. The values of the counters and the station header information can be displayed on the video terminal or printed on the hard-copy console device or on the line printer, whenever the operator enters the appropriate console command. Counters and

header information can also be output to the console or line printer automatically, under program control. Programs can also generate console messages to the operator.

The IET mode of data collection provides much more detailed information about the sequence of events in an experiment. Whenever an event of interest occurs (e.g., a response, a stimulus change, or simply the passage of a specified period of time), the following information is written in the dataset: a code for the class of event and the time since the last event was recorded. The IET data thus provides enough information to allow a complete reconstruction of the entire sequence of events in the experiment, with time resolution to 10 msec. As many as 64 distinct classes of events can be coded, with IETs allowed to range from 0.00 seconds to 7.7 days. If more than 64 classes of events are of interest, coding can be expanded to include storage of the value of a program variable with the event code. This expands greatly the detail with regard to events that can be recorded, but restricts the available range of IETs to about 6 hours--not a severe restriction for most behavioral purposes.

To summarize, the MANX system described above provides a means for the laboratory minicomputer (NOVA 800) to control and gather data from as many as 30 simultaneous but independent experiments. Each distinct experiment requires a different program, but the nature of the MANX programming language (State Logic Notation) makes it simple for experimenters to generate such programs. The number and variety of experiments that can be performed are not limited by the MANX software system, but rather by the hardware environment in which it operates.

The hardware environment in which MANX operates in this laboratory is diagrammed in Figure 2. The NOVA 800 minicomputer is an early (ca. 1962) model of Data General Corporation's NOVA line of computers. It is equipped with 32K words (16 bit) of core memory. The central processor is connected via controller boards to two mass storage devices (a dual disk drive and a 9-track magnetic tape unit), to a high-speed line printer, and (by switch selection) to either a teletype or a video terminal. Two digital I/O boards from G C Controls, Inc. of Smithville Flats, New York were recently added to the configuration. The I/O boards are installed in the computer main frame (Slots 11 and 14) and connected directly (via the backplane) to the NOVA processor. They provide 64 bits (lines) of digital input and 128 bits of digital output. The input bits require high-speed, 5-V transistor-transistor logic (TTL) signals to operate. The output bits provide TTL signals.

Each I/O board is connected via cables to one of the two MANX interface panels located in the laboratories. Each MANX interface panel (also from G C Controls, Inc.) has 32 input and 64 output circuits. The input circuits transform switch closures and other input signals into TTL signals and serve as buffers to protect the I/O boards and computer from laboratory voltages. The MANX panels also contain both switches for testing the input channels and indicator lights (light-emitting diodes, or LEDs) to indicate their status. The output circuits transform computer-produced signals (TTL) into higher power signals (24 V at up to 1.5 A) to drive laboratory apparatus that provide stimuli (lights, tones, aversive and appetitive reinforcers, etc.) to the subjects. Indicator LEDs on the MANX panel signal the experimenter when each output circuit is active.

The MANX interface panels are currently connected to three monkey-behavior and seven rat-behavior stations. Monkeys are being trained/tested on a short-term memory task (delayed matching to sample) with a secondary vigilance task. This set of tasks requires at least seven inputs and 20 outputs per station. Of the seven rat stations, six are used for relatively simple tasks (conditioned suppression or shuttle avoidance) which require only two inputs and six outputs per station. The remaining rat station is designed to gather shuttle avoidance data from nine rats simultaneously; it requires nine inputs and five outputs. With 89 of the 128 available outputs and 36 of the 64 available inputs currently occupied, MANX I/O capacity could handle an additional monkey station or several additional rat stations. Current experimental needs do not, however, require additional stations, since as many as 24 monkeys and 60 rats can be tested on a daily basis in the current configuration.

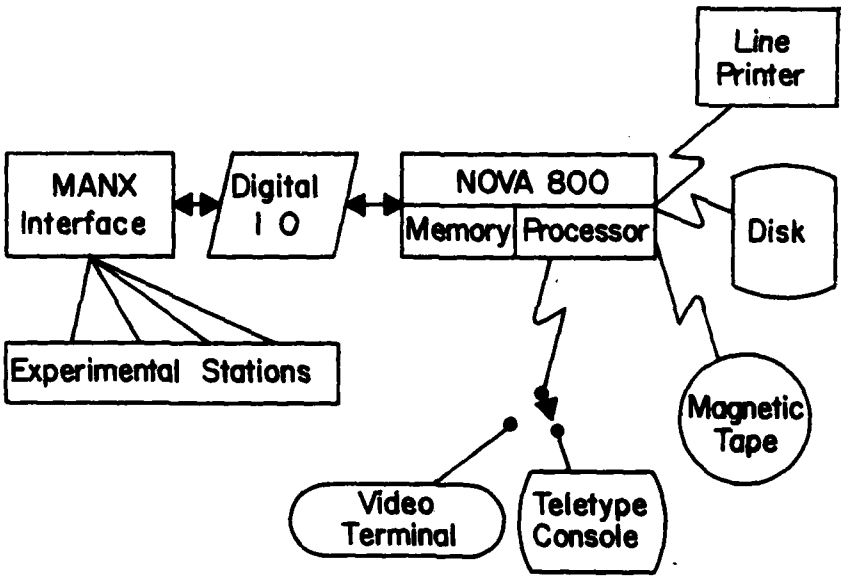


Figure 2. NOVA 800 -- MANX hardware system block diagram. The NOVA 800 uses MANX software (Fig. 1) to control experiments via digital I/O boards installed in the computer. Cables connect the I/O boards to MANX interface elements that condition input signals from and provide operational current to stimulus and response apparatus in the experimental stations. The operator enters commands from the teletype console on video terminal which causes the MANX RTS and MANX programs to be read into the computer from the disk. The RTS controls the stations and transfers data to files in the disk. Permanent copies of data files are made daily on magnetic tape. Both the line printer and the teletype console are used to provide immediate printed summaries of the results of each experimental run.

MANX PROGRAMMING IN STATE LOGIC NOTATION

The MANX software system allows an experimenter to translate a State Logic diagram directly into a machine language program to operate an experiment. State Logic developed from the mathematical theory of finite automata. Most readers will recognize it as represented by the familiar flow chart. It provides a description of sequential processes in which each output is completely and uniquely determined by the current input and the preceding set of inputs. The typical behavioral experiment is such a sequential process (12). For controlling such processes, State Logic has been shown to provide rules that involve the minimal amount of control logic while completely eliminating indeterminate states of the experiment or process (7). As a language for the control of behavioral experiments (13,14), State Logic Notation has several major advantages:

1) generality across different experimental problems;

2) simplicity, which makes it easy to learn and apply;

3) sufficient power to describe the most complex sequential processes and contingencies; and

4) efficient use of computer memory and processing capacities.

The applicability of State Logic Notation to behavioral experiments can be illustrated with a few simple examples. Figure 3 is a state diagram of the continuous reinforcement paradigm. In this paradigm, each response of the subject (e.g., a lever press, symbolized by R1) is followed immediately by the presentation of a reinforcing stimulus (e.g., the availability of food or water to the deprived animal, symbolized by SR). In Figure 3, the experiment has three states. State 1 represents the conditions before the experimenter starts the experiment. The animal is in a dark Skinner box; responses have no effect. When the experiment is started, transition (symbolized by an arrow) to State 2 occurs. At the instant of transition, the house light (HL) is turned on and a variable (A) is set to zero. In State 2, a response will produce transition to State 3, in which reinforcement is available for a limited period (4 s). Variable A is incremented by 1 each time a reinforced response occurs. At the end of 4 seconds in State 3, transition to either of the other two states can occur, depending on the value of variable A. If 100 reinforced responses have

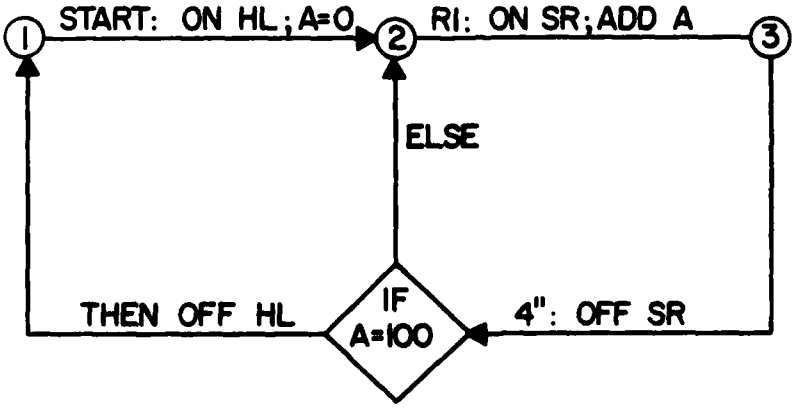


Figure 3. State diagram of program to produce continuous reinforcement of responses (R1) until 100 reinforced responses have occurred.

occurred, then transition is to State 1, otherwise (ELSE) transition is to State 2.

Figure 3 demonstrates nearly all the major features of State Logic Notation. More complex training or testing paradigms consist of more complex contingencies, but each such procedure can be reduced to a series of states that describe the momentary contingencies and the stimuli present while each state is active. Figure 3 describes a set of procedures, not the outcome of the experiment. If the animal never responds, no reinforcements will be delivered. If fewer than 100 reinforced responses occur, the house light will remain on.

While the State Logic diagram (Fig. 3) is a convenient and compact way of describing the contingencies in an experiment, it is not a convenient input form for a computer. It can, however, be readily translated into a MANX source program:

```
/ CRF-100           ("/" indicates that a comment will follow, which is
                    ignored by compiler)

/ PRGNO=1

/ Program to turn on house light at START,
/ provide 100 reinforced responses on continuous
/ reinforcement schedule, then turn off house light.

S.S.1,              / State set 1

S1,                 / First state, wait for START
   START:ON HL;SET A=0 --->  S2

S2,                 / Detect responses, start reinforcement
   R1:ADD A;ON SR --->  S3

S3,                 / End reinforcement, count reinforcements and
                    / test for 100 reinforced responses
   4" & A(100):OFF SR;OFF HL --->  S1   / IF A=100 THEN STOP
               :OFF SR --->  S2         / Else return

$                                        / End of Program
```

This simple MANX source program, after being entered into the computer via a text editor, can be compiled by the MANX compiler, which produces a binary (machine language) program. Whenever the binary program is loaded to a behavioral station under control of the MANX RTS, the set of state contingencies diagrammed in Figure 3 will be in effect for that behavioral station.

This MANX program consists of a single state set; i.e., a set of states related such that each state (except the first, which is always entered at the start of the program) can be entered by transition from at least one other state. More complex programs often have several state sets, each accomplishing a separate function. For example, one state set might detect responses and set up reinforcements under a complex set of contingencies, while another state set actually delivers the reinforcements and another constructs a frequency distribution of interresponse times (IRTs).

MANX provides a convenient device, called a Z-pulse, to allow separate state sets in a program to interact with each other. Note that each statement in the sample program presented above has the form

Condition:Operation(s) Transition to--> Destination State

In each state certain conditions are specified. Whenever an event occurs that satisfies the condition, transition to another state occurs. Operations on program-controlled devices and on program variables can occur only when the condition is met and transition occurs. Z-pulses are artificial events that can be generated as an operation when a transition takes place. They can also serve as an input to satisfy a condition and produce state transitions. To illustrate:

```
/ PROGNO=2 -- VR schedule with IRT distribution.
/ A three-state-set program to present reinforcements on a
/ variable ratio schedule, the ratio (number of required
/ responses) for each reinforcement being taken serially from
/ a LIST.  Counters provide a frequency distribution (bin
/ width = 2 s) of interresponse times.  Z1 pulse is used
/ to initiate presentation of reinforcement (3 s of dipper
/ availability) and to select the next ratio from the LIST.

S.S.1,              / State set to start and stop program and count
                    / responses

S1,                 / Wait for start, set up reinforcement for
                    / first response
   START: ON HL;Set A=1, B=10, C=3 ----> S2

S2,                 / Detect and count responses, stop after 50
                    / reinforcements
   R1:  Z1; C1 ----> SX                / null transition
   50Z2:OFF HL;Z3 ----> S1

S.S.2,              / Ratio contingency, operate and count
                    / reinforcements

S1,                 / Wait for variable number of responses, start
                    / reinforcement, select next ratio
   AZ1: LIST 10,B,A,10,5,15,3,12,20,4,16,8,17;C2;ON SR ----> S2

S2,                 / Stop reinforcement
   3": OFF SR;Z2 ----> S1

S.S.3,              / Frequency distribution for IRTs

S1,                 / Wait for first response
   Z1: ----> S2
```

```
S2,                  / Increment bin pointer, count and reset on
                     / response
   2":ADD C ----->  S2                    / Increment bin pointer every 2"
   Z1:CC; Set C=3 ----->  S2              / Count and reset bin pointer
   Z3: ----> S1                           / Return after 50 reinforcements

$                                         / End of Program
```

In this example, State Set 1 merely starts and stops the program and detects and counts (C1) responses. Each response produces a Z1 pulse that is used by the other state sets. State Set 2 waits for a variable number (the value of variable A) of responses, then starts and counts (C2) reinforcements and selects the next ratio value from the LIST. The end of reinforcement generates a Z2 pulse, which is used by State Set 1 to stop the program at the end of 50 reinforcements. State Set 3 waits for the first response and thereafter increments a counter on each response. The counter incremented (C) corresponds to a bin of a histogram of IRTs. Counter 1 contains the total number of responses; Counter 2, the number of reinforcements; Counters 3, 4,..., N contain frequencies of IRTs between 0 and 2 seconds, 2 and 4 seconds, ..., 2N-6 and 2N-4 seconds, respectively.

This program illustrates the use of Z pulses to communicate within and among state sets, as well as the use of program variables and LIST--one of a number of MANX system functions. Variable A was initially set to 1 so that the first response would be reinforced. Thereafter, A sequentially took on the values in the LIST function, which were selected to yield a variable ratio of responses to reinforcements with a mean value of 10 responses per reinforcement. Variable B is used by the LIST function as a pointer to the next value to be taken from the list. By convention it was initially set equal to the number of elements in the list. Variable C was used as a pointer to the counter array for the IRT frequency distribution. On each response, Counter number C was incremented; then C was set to 3. Until the next response, C was incremented every 2 seconds.

A state diagram (Fig. 4) may help to clarify the operation of this program. Note that a transition from a state to itself can occur in two ways. In State Set 1, the R1: transition from State 2 to State 2 is an example of the "null transition," symbolized by a broken arrow on the state diagram and by (----> SX) in the program example. The null transition allows reentry to the state without reinitializing any counters or timers (e.g., 5OZ2:). Thus, after entry from State 1, any number of response-produced null transitions may occur while the 5OZ2: contingency remains in effect. In State Set 3, the recurrent transitions from State 2 to State 2 have a different form. Transition to State 2 is indicated normally, so the timer for the 2-second contingency is reset on each reentry to the state.

Z pulses can also be used as part of a compound condition (logical AND function). For example:

```
S1,
   R1:Z1 ----> S2              (When R1 occurs generate Z1 pulse and go
                               to State 2)
```

```
S2,
   Z1 & P(500):ON1 ----> S3        (When Z1 occurs, with probability = .500,
              :ON2 ----> S4         turn on Stimulus 1 and go to State 3;
                                    otherwise, turn on Stimulus 2 and go to
                                    State 4)
```

Another example:

```
S1,        10":Z1 ----> S2        (After 10 s, branch to State 3, 4, or 5,
                                    depending on whether Variable S has the
S2,                                 value 0, 1, or 2.  If none of the above,
   Z1 & S(0): ----> S3              return and wait 10 s more)
         S(1): ----> S4
         S(2): ----> S5
             : ----> S1
```

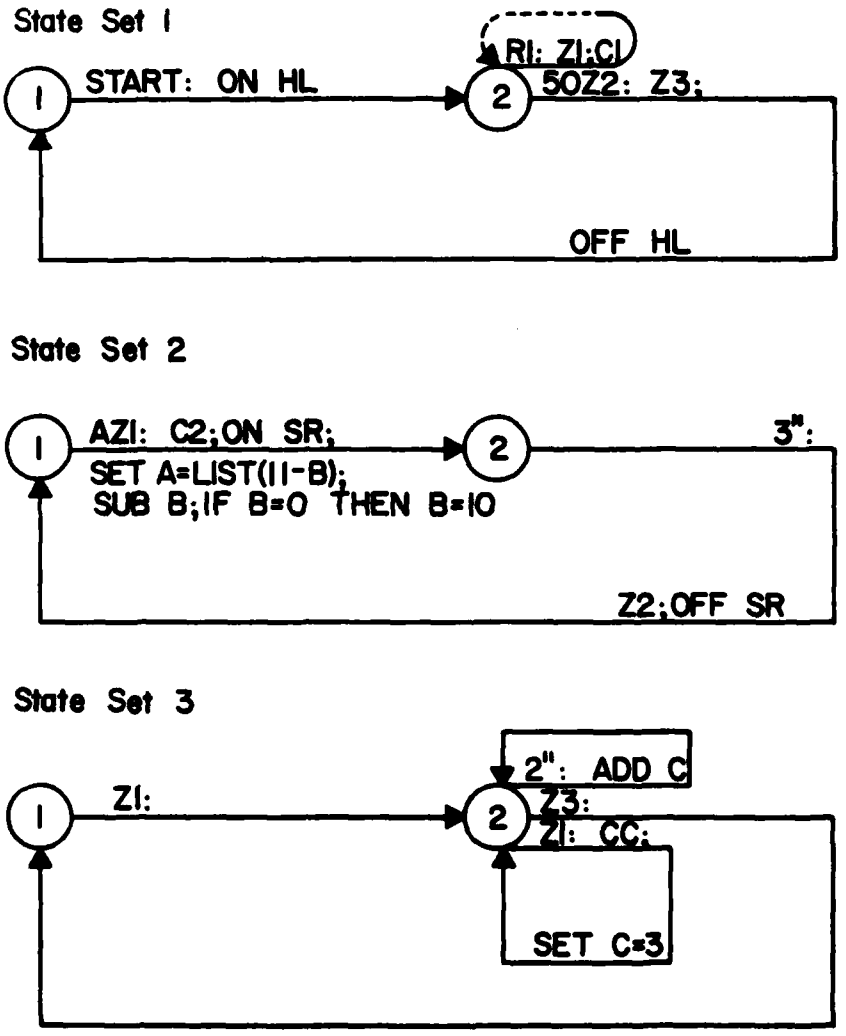


Figure 4. State diagram of variable ratio with interresponse time distribution.

This example illustrates some of the combinations of logical OR and AND functions available. Note that State 2 could also be entered from another state, and the state set might then remain in State 2 until another state set caused a Z1 pulse to be generated. Thus a single branch point can serve different functions in a program, depending on when and how transitions to it occur. Following is a brief glossary of conditions and operations frequently used in MANX programs.

Condition:	Satisfied by:
10R2:	Occurrence of 10 responses of class 2
RT:	Time elapsed since state entered = the value of Variable RT.
Z9:	The generation of a Z9 pulse, either by another state set or by the state from which transition occurred.
R1 & P(250):	R1 (with probability = .25).
Z2 & A(10):	The occurrence of a Z2 pulse if and only if the value of Variable A is currently 10.
AZ5:	The occurrence of a number of Z5 pulses, the number being determined by the value that Variable A had when the state was entered.

Operation:	Effect:
ON 2	Turn on Stimulus 2
OFF 9	Turn off Stimulus 9.
ON X	Turn on a stimulus selected by the current value of Variable X.
ADD X	Increment the value of Variable X.
SUB Y	Decrement the value of Variable Y.
SET A=99	Set Variable A = 99.
SET B=A-10	Set Variable B = 10 less than the current value of A (possible operators are +, -, /, and *, but result must always be a positive integer).
SET CT=120'	Set time variable C=2 hours.
CODE 12	Store Code 12 with the time since the last coded event.
CODE N	Store a code determined by the value of N with the time since the last coded event.
C9	Increment Counter 9.
CP	Increment the counter that Variable P points to.

TIME X, Y, Z Get the time of day in hours and minutes.

WRITE 60,X Store code 60 with value of X (e.g., time in hours, minutes).

CALL TEST(A,B,C,)
Set C = 1 if A > B
= 2 if A = B
= 3 if A < B

RAND N,A,B,N_1,N_2,N_3,...,N_N
Select randomly (without replacement) from the N values listed. Assign the value selected to Variable B.

TYPE "MESSAGE",N,V
Send MESSAGE and values of Variables N and V to system console.

LIST N,A,B,N_1,N_2,...N_N
Set Variable B = the Ath member of the list and increment A. If A > N, set A = 1.

A complete and rather complex functional program used to assess short-term memory in the rhesus monkey is presented for illustrative purposes in Appendix A. This program accomplishes a titrated delayed-match-to-sample task. On each trial, the program presents a randomly selected color (the sample) on a back-lighted response key. The monkey is required to indicate that he has seen the sample by touching the key, at which time the sample light is turned off. After a variable delay, three other keys are lighted with three different colors, one of which matches the sample. The monkey's task is to choose and touch the key illuminated with the color that matches the previously presented sample. The position of the correct match varies randomly from trial to trial. On each trial, the colors of the sample and the two incorrect alternatives are selected randomly from among four colors: red, green, blue, and white. The duration of the delay varies from trial to trial, depending on the monkey's performance on preceding trials. After any two consecutive errorless trials, the delay is increased by 1 second. After each trial in which an error occurs, the delay is reduced by 1 second. Within each test session, the delay will thus oscillate around a duration at which the animal can maintain 66.7% accuracy of performance. After a brief period of settling in at the beginning of the session, the average delay can be used as a direct measure of the short-term memory of the monkey.

Before monkeys can perform stably at this complex and difficult task, they require a lengthy training period, beginning with much simpler tasks and progressing to more complexity. As each animal masters each phase of the training, the task is gradually modified in closer and closer approximation to the final task. The MANX system is extremely well adapted for such a training program. MANX provides the experimenter with the flexibility necessary to allow each animal to progress at his own pace. The progress of each animal can be closely monitored, and programs can easily be modified to facilitate progress. Each phase of the training is represented by a different MANX program. The whole group of animals is not required to progress in lock-step through the training sequence because any animal can be trained using any MANX program in the sequence at any time.

To attempt such a training program using traditional hard-wired behavioral control apparatus would require heroic efforts, since the apparatus would require modification on a daily (or even hourly!) basis. The availability of a system like MANX not only reduces the effort required for such a training program to a small fraction of what would be required without MANX, it also enables a variety of other behavioral research projects to be conducted during this lengthy training program.

While 15 to 18 monkeys were being trained daily over a period of months, several experiments on the effects of ionizing radiation and psychological stress on avoidance behavior in the rat were carried out. The details of instrumentation and programming for all of these experiments will be fully documented in separate reports.

The purpose of this report has been to describe the general nature of MANX, an integrated hardware/software system that is easily adaptable to any investigation requiring precise control and measurement of animal behavior. The complex behavioral program presented in Appendix A will illustrate the power and flexibility of the MANX programming language as an implementation of State Notation. This programming system, coupled with the support software and hardware interfacing systems described in this report, facilitates optimal utilization of the NOVA laboratory computer for behavioral experimentation.

REFERENCES

1. Ferster, C. B., and B. F. Skinner. Schedules of reinforcement. New York: Appleton-Century-Crofts, 1957.

2. Gilbert, S. G. MANX introduction. Rochester, New York: MANX Systems, Incorporated, 1979.

3. Gilbert, S. G. MANX support software. Rochester, New York: MANX Systems, Incorporated, 1979.

4. Gilbert, S. G. MANX users manual. Rochester, New York: MANX Systems, Incorporated, 1979.

5. Gilbert, S. G., and D. C. Rice. NOVA SKED: A behavioral notation language for Data General minicomputers. Behav Res Meth Instrum 10:705-709 (1978).

6. Gilbert, S. G., and D. C. Rice. NOVA SKED II: A behavioral notation language utilizing the Data General Corporation real-time disk operating system. Behav Res Meth Instrum 11:71-73 (1979).

7. McClusky, E. J., Jr. Introduction to the theory of switching circuits. New York: McGraw-Hill, 1965.

8. Millenson, J. R. On-line sequential control of experiments by an automated contingency translator. In B. Weiss (ed). Digital computers in the behavioral laboratory. New York: Appleton-Century-Crofts, 1973.

9. Miller, G. A., A. S. Bregman, and D. A. Norman. The computer as a general purpose device for the control of psychological experiments. In R. W. Stacy and B. D. Waxman (eds). Computers in biomedical research, Vol. I. New York: Academic Press, 1965.

10. Polson, P. G. SCAT: Design criteria and software. Behav Res Meth Instrum 5:241-244 (1973).

11. Skinner, B. F. The behavior of organisms. New York: Appleton-Century-Crofts, 1938.

12. Snapper, A. G., and G. Inglis. SKED software system, manual 3, revision C. Time-shared SUPERSKED. Kalamazoo, Michigan: State Systems, Incorporated, 1979.

13. Snapper, A. G., and R. M. Kadden. Time-sharing in a small computer through the use of a behavioral notation system. In B. Weiss (ed). Digital computers in the behavioral laboratory. New York: Appleton-Century-Crofts, 1973.

14. Snapper, A. G., J. Knapp, and H. Kushner. Mathematical description of schedules of reinforcement. In W. N. Schoenfeld (ed). The theory of reinforcement schedules. New York: Appleton-Century-Crofts, 1970.

15. Stacy, R. W., and B. D. Waxman (eds). Computers in biomedical research, Vol. I. New York: Academic Press, 1965.

16. Weiss, B. (ed). Digital computers in the behavioral laboratory. New York: Appleton-Century-Crofts, 1973.

17. Wood, R. W., W. F. Sette, and B. Weiss. Interfacing the experimenter to the computer: languages for psychologists. Am Psychol 30:231-238 (1975).

APPENDIX A.

MANX SOURCE PROGRAM FOR TITRATED DELAYED-MATCH-TO-SAMPLE

(Delay between sample offset and match onset titrated so as to maintain 66.7% accuracy of response.)

```
/DMTS.SK          NUMBER OF COUNTERS = 36     PROGRAM NUMBER =45
/TITRATED DELAYED-MATCH-TO-SAMPLE, SAMPLE IS RANDOM SELECTION
/FROM 4 HUES (RED, GREEN, BLUE, WHITE). ONE ALTERNATIVE MATCHES
/SAMPLE, THE OTHER 2 ARE SELECTED RANDOMLY FROM REMAINING 3 HUES.
/AFTER ERROR, PRESENTATION IS REPEATED UNTIL TRIAL ENDS WHEN
/CORRECT ALTERNATIVE IS SELECTED WITHOUT ERROR.   POSITION
/OF CORRECT ALTERNATIVE CHANGES AFTER EACH ERRORLESS TRIAL
/(RANDOM 50-50 FROM THE 2 INCORRECT POSITIONS OF THE PRECEDING
/TRIAL).  DELAY STARTS AT 4", INCREASES 1" FOR EACH PAIR OF
/CONSECUTIVE ERRORLESS TRIALS, DECREASES 1" FOR EACH ERROR
/TRIAL.   COUNTERS ARE PRINTED AUTOMATICALLY AT END OF RUN.
/SHOCKS FOR SLOW RESPONSE AFTER 2", 2 HZ AT 10% DUTY CYCLE FOR
/3", THEN 20% DUTY CYCLE.   ERRORS PUNISHED BY 0.5" SHOCK.
/LATENCY HISTOGRAM BINS ARE .25" WIDE.

/STIMULUS CODES (SCODE)
/4 = RED
/5 = GREEN
/6 = BLUE
/7 = WHITE
/OUTPUTS (STIMULI)
/ 0 + SCODE - SAMPLE
/ 8 + SCODE - LEFT ALT.
/16 + SCODE - CENTER ALT.
/24 + SCODE - RIGHT ALT.
/17         - SHOCKER

/Z-PULSES
/Z1 - INTERNAL (S.S.1)
/Z2 - STOP SHOCK CYCLE
/Z3 - START SHOCK CYCLE
/Z4 - TURN ON SHOCK
/Z5 - CORRECT RESPONSE
/Z6 - ERRORLESS TRIAL
/Z7 - R2 CORRECT
/Z8 - R3 CORRECT
/Z9 - R4 CORRECT
/Z10- SAMPLE ON
/Z11- INTERNAL (S.S.7)
/Z13- ERROR RESPONSE
/
/COUNTERS
/1-15 = SAMPLE R LATENCY
/16   = LATENCY > 3.75"
/17-31= MATCH R LATENCY
/32   = LATENCY > 3.75"
/33   = ERRORLESS TRIALS
/34   = ERROR TRIALS
/35   = TOTAL ERRORS
/36   = TOTAL SHOCKS
/
/
/
/

/INPUTS
R1 - SAMPLE KEY
R2 - LEFT MATCH KEY
R3 - CENTER MATCH KEY
R4 - RIGHT MATCH KEY
R5 - MANUAL SHOCK OFF
R6 - END OF SESSION, STORE DATA
R17- SESSION START
R18- SESSION END

/VARIABLES
A = SAMPLE STIMULUS
B = INCORRECT HUE
C = INDEX FOR RAND. ITI CHOICE
D = CODE--CORRECT RESP. POS'N.
E = STATION NUMBER
FT= INTERTRIAL INTERVAL (ITI)
G = SELECTED (RANDOM) HUE
H = INDEX FOR RAND. HUE CHOICE
I = INCORRECT MATCH (1)
J = CORRECT MATCH
K = INCORRECT MATCH (2)
L = ERROR FLAG
MT= STORED DELAY VALUE
N = FLAG FOR SHOCK INCREASE
O = CORRECT RESPONSE CODE
PT= DELAY
Q = LEFT ERROR CODE
R = CENTER ERROR CODE
S = RIGHT ERROR CODE
T = FLAG FOR END OF TEST
U = HISTO BIN FOR RESP. LATENCY
V =
W = TEST FOR 2 ERRORLESS TRIALS
X = TIME, TEMP HUE VALUE
Y = TIME, DELAY TEST VALUES
Z = TIME
```

```
/KEYBOARD R17 STARTS PROGRAM, R18 STOPS IT.
/R6 SWITCH ALSO STOPS PROGRAM.
/CODES AS PER LABEL.DC

S.S.1,          /SET UP STIMULI, RUN TRIALS
S1,             /INITIALIZE
      .01": SET A=4,C=5,D=4,H=4,L=0,T=0,U=1;
      CALL STANO (17,E,Y,Z)---->S2
S2,             /START, ZERO IET CLOCK, STORE START TIME
      R17: TIME X,Y,Z; WRITE 62,X; CODE 64; Z5---->S3
S3,             /NEW TRIAL OR REPEAT
      Z5 & L(0): C33; Z6; Z1---->S4
               : C34; Z1---->S20        /ERROR, CORRECTION TRIAL
S4,             /IF NOT DONE, SELECT ITI
      Z1 & T(3): ---->S2
               : RAND5,C,FT,5",6",7",8",9"; Z1---->S5
S5,             /SELECT COLORS
      Z1 & T(3): ---->S2
               : RAND4,H,B,4,5,6,7; Z1---->S6
S6,             /ASSIGN COLORS
      Z1 & H(3): SET A=B; Z1---->S5
         & H(2): SET X=B; Z1---->S5
         & H(1): SET Y=B,H=4; Z1---->S7
               : TYPE "OOPS!"; SET H=4---->S5
S7,             /BRANCH ON LAST CORRECT POSITION
      Z1 & D(2): Z1---->S8
         & D(3): Z1---->S9
         & D(4): Z1---->S10
               : SET D=4,H=4; TYPE "OOPS1!"---->S5
S8,             /LAST WAS LEFT
      Z1 & P(500): SET D=3; Z1---->S12
                 : SET D=4; Z1---->S13
S9,             /LAST WAS CENTER
      Z1 & P(500): SET D=2; Z1---->S11
                 : SET D=4; Z1---->S13
S10,            /LAST WAS RIGHT
      Z1 & P(500): SET D=2; Z1---->S11
                 : SET D=3; Z1---->S12
S11,
      Z1: SET I=X+16,J=A+8,K=Y+24,O=A+12,R=I-14,S=K-17; Z1-->S14
S12,
      Z1: SET I=X+8,J=A+16,K=Y+24,O=A+16,Q=I-11,S=K-17; Z1-->S14
S13,
      Z1: SET I=X+8,J=A+24,K=Y+16,O=A+20,Q=I-11,R=K-14; Z1-->S14
S14,            /START NEW TRIAL OR QUIT
      Z1 & T(3): ---->S2
      FT: ON A; CODE 44; SET PT=MT; Z10---->S15

S15,            /DETECT EARLY SAMPLE R OR START SHOCK
      R1: OFF A; CU; CODE 45; Z1---->S17
      2": Z3; CODE 40---->S16
S16,            /DETECT LATE SAMPLE R, TURN OFF SHOCK
      R1: OFF A; CU; Z2; CODE 46; Z1---->S17
```

```
S17,              /TURN ON MATCHES
        PT: ON I; ON J; ON K; CODE 47; Z1---->S18
S18,              /SET UP RESPONSE CONTINGENCIES
        Z1 & D(2): Z7---->S19
           & D(3): Z8---->S19
           & D(4): Z9---->S19
                : OFF I; OFF J; OFF K; TYPE "OOPS2!"---->S2
S19,              /WAIT FOR RESPONSE OR TURN ON SHOCK
        Z5: Z5---->S3
        2": Z3---->S3
S20,              /START CORRECTION TRIAL
        2": ON A; CODE 48; Z10---->S15

S. S. 2,          /CORRECT RESPONSE AND ERROR DETECTION & CODING
S1,               /BRANCH ON CORRECT POSITION
        Z7: SET L=0---->S2
        Z8: SET L=0---->S4
        Z9: SET L=0---->S3

S2,               /R2 CORRECT (LEFT)
        R2: OFF I; OFF J; OFF K; CU; CODE 0; Z2; Z5---->S1
        R3: C35; CODE R; SET L=1; Z13---->SX
        R4: C35; CODE S; SET L=1; Z13---->SX

S3,               /R4 CORRECT (RIGHT)
        R4: OFF I; OFFJ; OFF K; CU; CODE 0; Z2; Z5---->S1
        R3: C35; CODE R; SET L=1; Z13---->SX
        R2: C35; CODE Q; SET L=1; Z13---->SX

S4,               /R3 CORRECT (CENTER)
        R3: OFF I; OFF J; OFF K; CU; CODE 0; Z2; Z5---->S1
        R2: C35; CODE Q; SET L=1; Z13---->SX
        R4: C35; CODE S; SET L=1; Z13---->SX

S. S. 3,          /RESPONSE DELAY SHOCKER CONTROL
S1,
        Z3: SET N=6; Z4---->S2
S2,
        Z4: ON 17; C36---->S3
S3,
        Z4 & N(0): TYPE "Z", E; SET N=20---->S5
        Z2: OFF 17---->S1
        .05": OFF 17; SUB N---->S4
S4,
        Z2: OFF 17---->S1
        .45": ON 17; C36; Z4---->S3
S5,                    /INCREASE SHOCK DURATION
        Z2: OFF 17---->S1
        .10": SUB N; Z1; OFF17---->S6
S6,
        Z1 & N(0): TYPE "L", E; SET N=20---->S7
                : ---->S7
S7,
        Z2: OFF 17---->S1
        .40": ON 17; C36---->S5
```

```
S. S. 4,            /SESSION STOP AFTER 50 ERRORLESS TRIALS
                          /AND/OR R18 OR R6

S1,
        R6: TIME X, Y, Z; WRITE 63, X---->STOP
        R5: Z2---->SX                /MANUAL SHOCK OFF
        R18: TIME X, Y, Z; WRITE 63, X; ---->STOP
/END OF SESSION
        51Z6: TIME X, Y, Z; WRITE 63, X; ON 1; SET T=3;
        CALL PTCNT(18, 1);
        TYPE "PROGRAM DONE, R18", E, "!"---->S2
S2,
        R18: DUMP---->STOP
        R6: DUMP---->STOP

S. S. 5,            /HISTOGRAM BIN COUNTING

S1,
        Z7: SET U=17---->S2
        Z8: SET U=17---->S2
        Z9: SET U=17---->S2
        Z10: SET U=1---->S2

S2,             /BINS FOR RESPONSE LATENCY
        Z2: ---->S1
        .25": ADD U; Z1---->S3

S3,             /LATENCIES > 3.75"
        Z2: ---->S1
        Z1 & U(16): ---->S1
            & U(32): ---->S1
                : ---->S2

S. S. 6,            /SHOCK FOR WRONG RESPONSE
S1,
        Z13: CODE 41; ON 17; C36---->S2
S2,
        .50": OFF 17---->S1

S. S. 7,            /TITRATE DELAY
S1,             /SET TEST VALUE AND INITIAL DELAY
        .01": SET W=1, MT=4"---->S2
S2,             /BRANCH ON ERROR, INCREASE IF LAST 2 CORRECT
        Z13: Z11---->S3                  /ERROR
        Z6 & W(2): SET MT=MT+1", W=1---->S2      /INCREMENT
                : ADD W---->S2                   /NEED ONE MORE
S3,             /CHECK FOR MINIMUM DELAY
        Z11: CALL TEST (11, MT, 2", Y); Z11---->S4
S4,             /DECREMENT IF NOT MIN.
        Z11 & Y(2): ---->S5              /MINIMUM
                : SET MT=MT-1"---->S5    /DECREMENT
S5,                     /WAIT FOR END OF TRIAL
        Z6: SET W=1---->S2       /RESET TEST VALUE AND RETURN

$       /END OF PROGRAM
```